Islamic University of Technology (IUT)

Organization of Islamic Cooperation (OIC)
Department of Electrical and Electronic Engineering (EEE)

Course No.: EEE 4606
Course Name: Microcontroller Based System Design Lab

Project Name: Designing a calculator using 8051 Microcontroller
Submission Date: 17.10.19

Section: C1
Group: 1
Group Members:

- Md Moinul Islam Khan 160021011
- Sabiha Sharmin 160021047
- Sabbir Ahmed 160021059
- Sanjida Ali 160021057
- Mohibul Islam 160021077
- Jubair Alam 160021085
- Fariha Mehjabin 160021095
- Tasnim Zaman Adry 160021157

Objective

To design a calculator which can add, subtract (able to show negative number), multiply and divide the two inputs which will be given from the keypad and result will be displayed in the LCD display.

Introduction

The 8051 Microcontroller is one of the most popular and most commonly used microcontrollers in various fields like embedded systems, consumer electronics, automobiles, etc. Technically called as Intel MCS-51 Architecture, the 8051 microcontroller series was developed by Intel in the year 1980 and were very popular in the 80's (still are popular). 8051 Microcontroller has many features like Serial Communication, Timers, Interrupts, etc. Even though 8051 Microcontroller might seem a little bit out of fashion, we feel that it is one of the best platforms to get started with Microcontrollers, Embedded Systems and Programming (both C and Assembly).

As we know 8051 is an 8-bit microcontroller which has 4KB of ROM and 128 Bytes of RAM. By this microcontroller we can do many kinds of programmable projects. Such as password access control, motor control, real time clock (RTC) etc. Here in this project we have made a calculator using this 8051 microcontroller. Here we have done simple arithmetic operations like addition, subtraction, multiplication and division. Primarily, the user will provide the numbers and the respective arithmetic operation. When the equal sign is pressed, then the desired result will be shown as output in the LCD display. As we have 8-bit registers in the microcontroller 8051, we can only perform operations which do not exceed 255 in the result.

The coding was done in MIDE-51 and it was built in it to form a corresponding .hex file. Then we implemented the circuit in Proteus software. We uploaded the .hex file in the microcontroller. The whole program was burnt into the 8051 kit.

Circuit in Proteus

The given figure shows the full circuit of our project. This circuit consists of a few components which are mentioned below:

- AT89C51
- BUTTON
- CAPACITOR
- CRYSTAL
- LM016L
- RESISTOR
- RESPACK-8

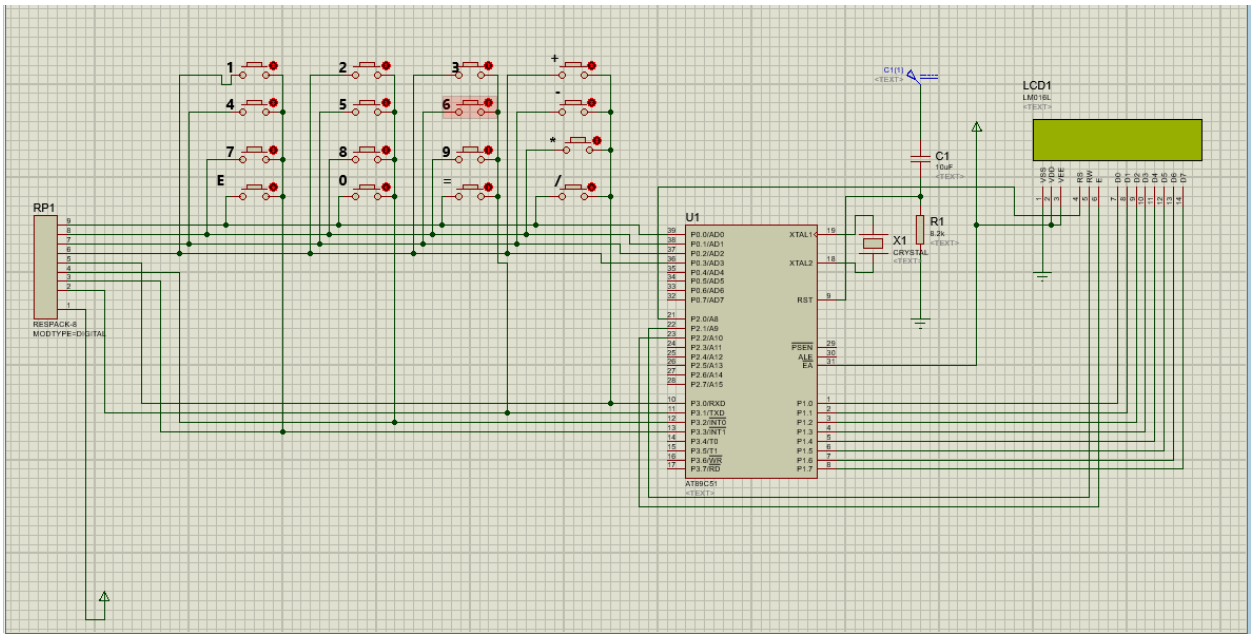

Figure: Circuit diagram

As we can see in the figure, the LCD data pins D0-D7 is connected with Port-1 of microcontroller. VSS is grounded. VDD and VEE are connected with EA pin of microcontroller. And the RS, RW and E pins of LCD are connected to pins P2.0, P2.1 and P2.2 respectively for initializing the LCD. Pins 18 and 19 of

microcontroller are connected with the crystal oscillator. The RST pin has been provided with a 5V DC. A 4x4 keypad has been connected with the microcontroller 8051.

```
ORG 00H
START: LCALL MEM_CLEAR
MOV R0,#50H
MOV R1,#60H
MOV R6,#00H
MOV R7,#00H
MOV A,#38H
LCALL COMNWRT
MOV A,#0EH
LCALL COMNWRT
MOV A,#01H
LCALL COMNWRT
MOV P3,#0FFH
MOV R0,#50H

K1:  MOV P0,#0
     MOV A,P3
     ANL A,#00001111B
     CJNE A,#00001111B,K1

K2:  LCALL DELAY
     MOV A,P3
     ANL A,#00001111B
     CJNE A,#00001111B,OVER
     SJMP K2


OVER:LCALL DELAY
      MOV A,P3
      ANL A,#00001111B
      CJNE A,#00001111B,OVER1
      SJMP K2

OVER1: MOV P0,#11111110B
      MOV A,P3
      ANL A,#00001111B
      CJNE A,#00001111B,ROW_0
      MOV P0,#11111101B
      MOV A,P3
      ANL A,#00001111B
      CJNE A,#00001111B,ROW_1
      MOV P0,#11111011B
      MOV A,P3
      ANL A,#00001111B
      CJNE A,#00001111B,ROW_2
      MOV P0,#11110111B
      MOV A,P3
      ANL A,#00001111B
      CJNE A,#00001111B,ROW_3
      LJMP K2
```

```
ROW_0:MOV DPTR,#KCODE0
      SJMP FIND
ROW_1:MOV DPTR,#KCODE1
      SJMP FIND
ROW_2:MOV DPTR,#KCODE2
      SJMP FIND
ROW_3:MOV DPTR,#KCODE3
      SJMP FIND

FIND:RRC A
      JNC MATCH
      INC DPTR
      SJMP FIND

MATCH:CLR A
      MOVC A,@A+DPTR
      LCALL DATAWRT
      INC R7
      LCALL CHECK
      ;LCALL DELAY2
      LJMP K1

CHECK: CJNE A,#'+',NOT_PLUS
         MOV 70H,A
         LCALL CONVERT
         LJMP K1
```

Keyboards are organized in a matrix of rows and columns. The CPU accesses both rows and columns through ports. Therefore, with two 8 bit ports an 8x8 matrix of keys can be connected to a microcontroller. When a key is pressed a row and a column make a contact. Otherwise there is no connection between rows and columns.

A 4x4 matrix connected to two ports is shown in the figure above. The rows are connected to an output port and the columns are connected to an input port.

It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed. To detect a pressed key the microcontroller ground all rows by providing zero to the output latch, and then it reads the columns. If the data read from columns is D3-D0 = 1111, no key has been pressed and the process continues till key press is detected. If one of the column bits has a zero, this means that a key pressed has occurred.

```
NOT_PLUS: CJNE A,#'-',NOT_MINUS
        MOV 70H,A
        LCALL CONVERT
        LJMP K1

NOT_MINUS: CJNE A,#'*',NOT_MUL
        MOV 70H,A
        LCALL CONVERT
        LJMP K1

NOT_MUL: CJNE A,#'/',NOT_DIV
        MOV 70H,A
        LCALL CONVERT
        LJMP K1

NOT_DIV: CJNE A,#'=', NOT_EQ

        ;CJNE 70H,#00,L1
        LCALL CONVERT
        LCALL EQUAL
        LJMP K1

NOT_EQ: CJNE A,#'E',STORE
        LCALL CLEAR
        LCALL MEM_CLEAR
        LJMP K1


STORE:
        CLR C
        SUBB A,#30H
        MOV R6,A

        MOV A,50H
        MOV B,#10D
        MUL AB
        MOV 50H,A
        MOV A,51H
        MOV B,#10D
        MUL AB
        MOV 51H,A
        MOV A,52H
        MOV B,#10D
        MUL AB
        MOV 52H,A
        MOV A,R6
        MOV @R0,A
        INC R0
        CLR A
        RET
```

```
CONVERT: CLR C
         MOV A,50H
         ADD A,51H
         ADDC A,52H
         MOV @R1,A
         INC R1
         LCALL MEM_CLR
         MOV R0,#50H
         RET

EQUAL:
        CLR C
        MOV A,70H
        CJNE A,#'+',MINUS
        MOV A,60H
        MOV B,61H
        ADD A,B
        LCALL DISPLAY
        LJMP K1

MINUS: CLR C
        CJNE A,#'-',MULTIPLICATION
        MOV A,60H
        MOV B,61H
        CLR C
        CJNE A,B,MIN_CHK


        MIN_POS:MOV A,60H
                MOV B,61H

        MIN_NEG:
                CLR C
                SUBB A,B
                LCALL DISPLAY
                LJMP K1

MIN_CHK: JNC MIN_POS
        MOV A,#'-'
        LCALL DATAWRT
        MOV A,61H
        MOV B,60H
        SJMP MIN_NEG

MULTIPLICATION: CLR C
        CJNE A,#'*',DIVISION
        MOV A,60H
        MOV B,61H
        MUL AB
        LCALL DISPLAY
        LJMP K1
```

```
DIVISION: CLR C
        MOV A,60H
        MOV B,61H
        DIV AB
        LCALL DISPLAY
        LJMP K1


DISPLAY:MOV B,#10D
        DIV AB
        MOV R7,B
        MOV B,#10D
        DIV AB
        ADD A,#30H
        LCALL DATAWRT
        MOV A,B
        ADD A,#30H
        LCALL DATAWRT
        MOV A,R7
        ADD A,#30H
        LCALL DATAWRT
        RET


CLEAR: MOV A,#1
        LCALL COMNWRT
        LJMP START

MEM_CLEAR: MOV 50H,#00H
        MOV 51H,#00H
        MOV 52H,#00H
        MOV 53H,#00H
        MOV 60H,#00H
        MOV 61H,#00H
        MOV 62H,#00H
        MOV 70H,#00H
        RET

MEM_CLR: MOV 50H,#00H
        MOV 51H,#00H
        MOV 52H,#00H
        MOV 53H,#00H
        RET

COMNWRT:MOV P1,A
        CLR P2.0
        CLR P2.1
        SETB P2.2
        LCALL DELAY2
        CLR P2.2
        RET
```

```
DATAWRT:MOV P1,A
        SETB P2.0
        CLR P2.1
        SETB P2.2
        LCALL DELAY2
        CLR P2.2
        RET
DELAY:
        PUSH 3
        PUSH 4
        PUSH 5
        MOV R3,#25
      L3:MOV R4,#4
      L2:MOV R5,#98
      L1:DJNZ R5,L1
         DJNZ R4,L2
         DJNZ R3,L3
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         POP 5
         POP 4
         POP 3
         RET
DELAY2:
        PUSH 6
        PUSH 7
        MOV R6,#50
       A1:MOV R7,#255
       A2:DJNZ R7,A2
          DJNZ R6,A1
          POP 7
          POP 6
          RET

KCODE0: DB '/','=','0','E'
KCODE1: DB '*','9','8','7'
KCODE2: DB '-','6','5','4'
KCODE3: DB '+','3','2','1'
        END
```

We at first allocated memory locations for the inputs given by the user. So both the numbers and the operation to be done are stored in different registers at first. Then we checked whether the user wants addition, subtraction, multiplication and division. If none of these operations are asked, then either the operation is ‘equal’ or ‘clear memory’.

According to the operation given, the subroutines of Addition, Subtraction, Multiplication and Division are called. Then after performing the operation, the result is saved in another memory location. Then the result is converted into the corresponding ASCII and is shown in the LCD.

Outputs

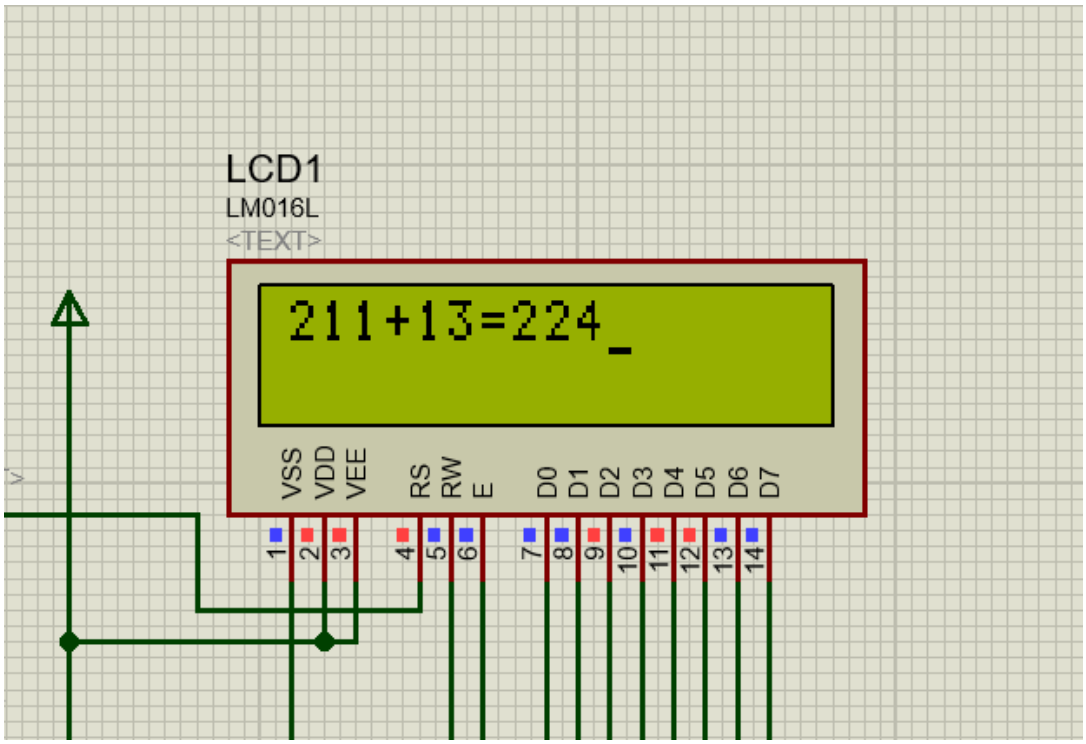

Figure: Addition operation

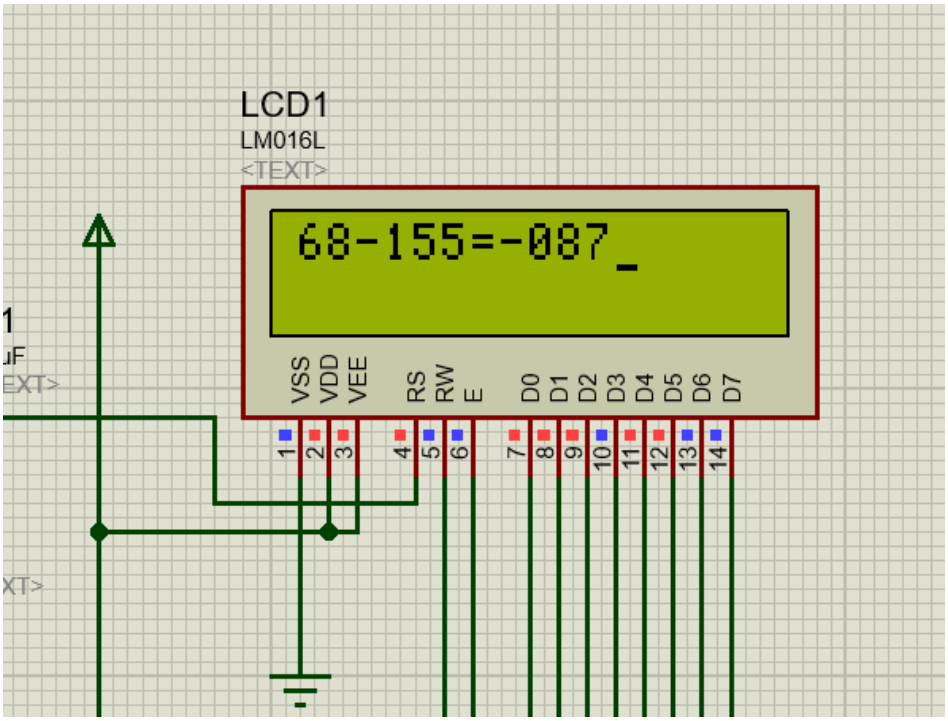

Figure: Subtraction operation

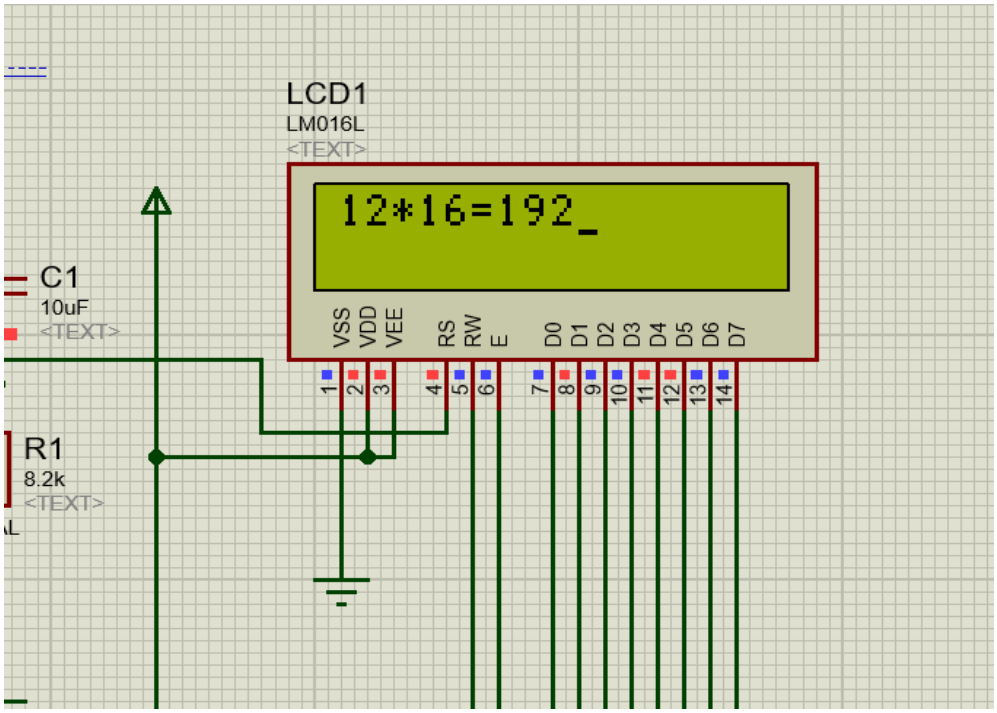

Figure: Multiplication operation

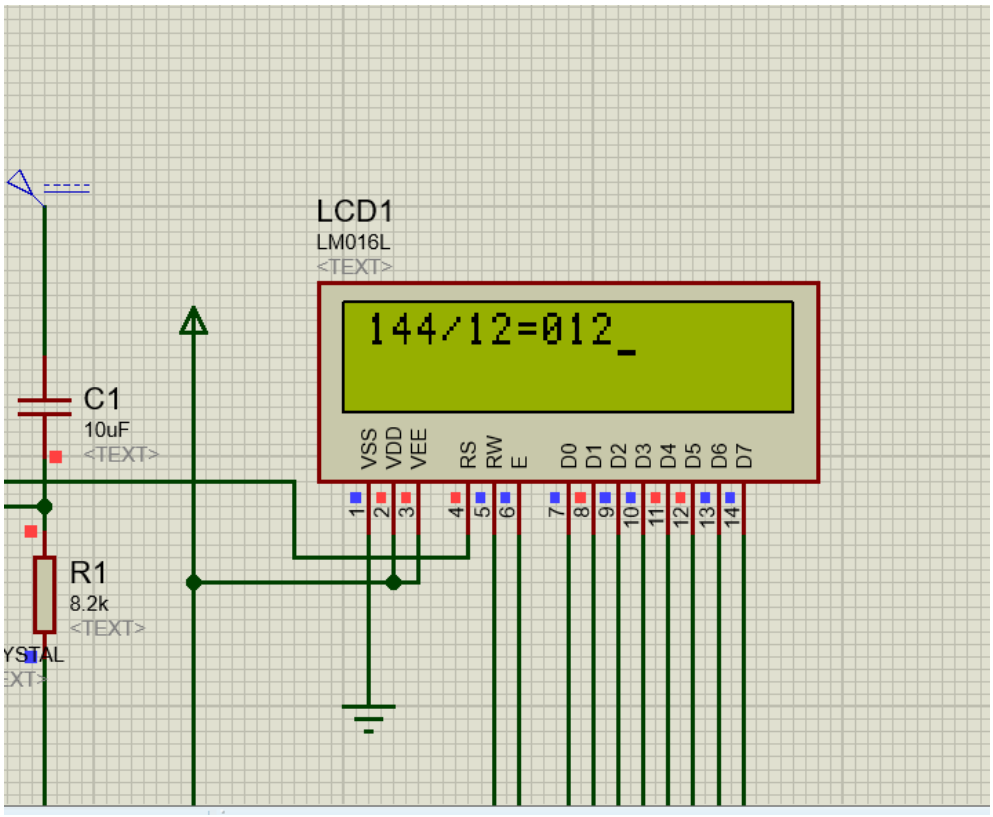

Figure: Division operation

Conclusion

We can perform all the operations with our calculator for 3 digits. But when the inputs exceed 4 digits, then only the division operation doesn't work. And when a fractional number comes as output after division, then we can see the quotient as output in LCD only, not the remainder. In case of 4 digits, the result comes correctly in hexadecimal format stored in multiple memory locations. Just the conversion from hexadecimal to decimal could not be done correctly.